Reproducing DNS 10Gbps flooding attacks with commodity-hardware

Santiago Ruano Rincón*, Sandrine Vaton*, Stéphane Bortzmeyer†
*Institut Mines-Télécom - Télécom Bretagne - 29238 Brest Cedex 3 - France
†AFNIC - Immeuble International - 78181 Saint-Quentin-en-Yvelines - France
Email:{santiago.ruano-rincon,sandrine.vaton}@telecom-bretagne.eu, bortzmeyer@nic.fr

***Abstract*—Being DNS an essential service for Internet reliability, it is an attractive target for malicious users. The constantly increasing Internet traffic rate challenges DNS services and their attack detection methods to handle actual queries while being flooded by tens of millions of malicious requests per second. Moreover, state of the art on hostile actions evolve fast. DNS administrators continuously face new kinds of attacks and they regularly need to evaluate their detection systems. We have studied different approaches to develop a tool able to reproduce state-of-the-art attacks, aiming to make it easy to evaluate countermeasure strategies. We have focused on commodity-hardware, DPDK and MoonGen to build a flexible flood query generator. The described tool can saturate a 10Gbps link, sending more than 12 million attack-like random DNS requests per second.**

I. Introduction

It is well-know that DNS is an Internet critical service. Internet architecture makes high-level services require DNS to identify resources associated to specific domain names. Out-of-service DNS servers easily yield to partially blackout Internet, and low-performance DNS servers slow down the rest of network activity depending on them. It is then an attractive target to act against Internet resilience, especially at nation-wide context when random qnames attacks address a specific country Top Level Domain (TLD) [1]. Moreover, DNS servers have been susceptible to be used as attacking sources, via reflect-and-amplify methods [2]. To that extend, DNS administrators need monitoring tools that help to detect and classify state-of-the-art flooding attacks, and to identify and understand zero-day vulnerabilities.

Also, increasing traffic challenges the current infrastructure, and it is common that detection systems cannot hold the CPU-intensive analysis under actual flooding, and even crash before the server they protect does so [3].

In this paper we study different supports to build a DNS flooding generator. In Section II we describe the state of the art on attacks against DNS and currently identified countermeasures. Considering that flexibility is an important requirement, we have chosen to rely on commodity-hardware and packet dispatching controlled by software. Then, in Section III we study different commodity-hardware network frameworks, and in Section IV, different existing traffic generators relying on them.

In Section V, we describe how we have built a DNS flooding tool based on the MIT-Licensed MoonGen [4], that focuses on Lua scripting and the underlying Data Plane Development Kit (DPDK) technology. Finally, we evaluate our tool's performance in Section VI.

The developed generator represents the first element on a testbed aimed to analyse countermeasure strategies against DNS flooding attacks. As we describe in the paper, using a single CPU core, this tool is able to saturate a 10Gbps link at wire-rate with structured DNS ∼74-byte packets, similar to random qnames flooding or reflect-and-amplify queries. DNS administrators might use this generator to stress their attack detection systems and evaluate countermeasure strategies. French legislation, mainly the 323-3-1 article from the Penal code, makes it legally impossible to openly publish software especially able to act against computer infrastructure, unless legitimate reasons such as research on security. Nevertheless, the DNS protocol support that we were required to create for MoonGen is currently available in its upstream sources. We invite interested security researchers to contact us to gain access to the full code.

II. State of the art on DNS flooding attacks and countermeasures

DNS servers face different kinds of flooding attacks today. In this paper, we focus on two of the most important: random qnames and reflect-and-amplify. In this section, we characterise them and describe some of the currently identified countermeasures.

A. Random qnames attack

The Distributed Denial of Service (DDoS) random qnames are commonly used to attempt against the availability of authoritative servers of a specific domain. As its name suggests, the attacker floods the server with queries composed of non-existent random prefixes and a fixed domain suffix found under the servers' authority. As it is common in DDoS, this attack requires the offender to control and coordinate a high number of DNS clients able to send packets forging their source IP addresses.

AFNIC reports a random qnames attack on September 4 2014, addressed against the Wallis-et-Futuna's .wf domain, that shares the TLD servers with France [5]. This attack lasted for fourteen hours, reaching a maximum rate of one million requests per second. DNSMON, a DNS active measurement system [6] from the RIPE Atlas network, reports more than 99% unanswered queries from the *e.ext.nic.fr* server, between

11:00 and 13:10, during this attack. Offenders were then able to partially made unavailable one of the AFNIC TLD servers.

Another random qname example is the big attack against the DNS root in November/December 2015 [7]. Note also that many DDoS attacks are not discussed or even announced publicly, such as the attack against RIPE-NCC in January 2016, whose technical report made at a DNS-OARC meeting is not public. These attacks could imply unusual packet characteristics, such as transmission on TCP or IPv6, but packets present the following general pattern.

```
09:12:34.802102 IP (tos 0x0, ttl 48, id 11149, offset
 0, flags [none], proto UDP (17),
length 86) 74.125.43.82.48819 > 194.0.9.1.53: 12752
[1au]
A? abwvgxmfktotuh.www.dafa888.wf. (58)
```

Generating a synthetic random qnames attack requires then to consider the following packet characteristics: (1) Varying source IPv4 or IPv6 addresses, (2) DNS server's IP as fixed destination, (3) varying query types, i.e., A, AAAA, ANY or TXT, and (4) varying queried qname over a fixed domain suffix, such as *randomqname.example8888.com*.

B. Reflect-and-amplify attack

The second type of attack we focus on is reflect-and-amplify, whose strategy is to make use of DNS servers to flood a target [2]. In this case, the attacker control several machines, making them to spoof their IP addresses with the victim's and to query a DNS server. Looking to produce from it the largest possible packet answer, it is common that the attacker takes advantage of flaws in the design of the protocol stack and of later extensions. For example, the DNS extension EDNS(0) [8], defined to overcome the size restrictions of DNS messages when sent over UDP, and that was required to exchange large DNSSEC records.

Two kind of DNS servers are commonly concerned to reflect the DNS messages: open resolvers and TLD servers. When using open resolvers, the attacker usually controls a domain where it includes large resources, specially of type TXT. TLD servers use to have better resources so they are better reflectors, but in general, they are better protected so it is more difficult to use them. To take advantage of TLDs and look for the largest amplification as possible, the attacker might request ANY type of resources, especially on DNSSEC signed domains, and advertise large UDP buffer size in EDNS's OPT-pseudo records. As we explain latter, the Shield of Perseus (SOP) generator [2] is able to send well-structured reflect-and-amplify queries, such as the following:

```
09:39:38.229993 IP (tos 0x0, ttl 20, id 16640,
   offset 0, flags [none], proto UDP (17), length 68)
 192.168.24.1.mdns > 192.168.24.2.domain:
   [udp sum ok] 55309+ [1au] ANY? example.com. ar: .
   OPT UDPsize=9000 OK (40)
```

Reproducing reflect-and-amplify attacks needs to take into account following packet characteristics: (1) Fixed victim's spoofed address as source IP; (2) public open DNS resolvers or TLD servers IP addresses; (3) commonly, query type ANY; (4) fixed or randomized qname; (5) include an EDNS0 additional record advertising a large buffer size, e.g. 9000 bytes as in most attacks in 2011-2012, a value which never appears in real requests, where the typical buffer size is 4096 bytes; (6) indicate the resolver handles DNSSEC security records.

The design of a flexible DNS generator requires to allow the user to set up different packet characteristics, including fixed and varying fields. Although, it should not be limited to the attacks that we have described here, but rather make it easy to take into account other scenarios, packet fields and further DNS extensions.

C. DNS monitoring and measures against flooding attacks

Besides the common and classic Nagios [9], there are free software tools that can be used by DNS administrators, such as the Ripe Atlas Network DNSMON [6], or the components of DNSwitness, DNS delve and DNSmezzo [10].

DNSMON and DNSdelve are active measurement tools. DNSMON uses the RIPE Atlas measurement network to provide an up-to-date service overview of DNS root and different TLD servers. It continuously measures the performance between RIPE anchor probes and the DNS servers. DNSMON makes it possible to query historical data, as the status of .fr servers during the random qnames attack described above. DNSdelve allows the user to ask DNS servers explicit questions related to the content of the zone, such as: "how many domains have at least an IPv6 Web server?" or "how many domains have SPF enabled?"

In contrast, DNSmezzo is a framework to capture and analyse DNS packets and perform passive measurement. DNSmezzo stores data in relational databases to allow long-term surveys. It makes it possible to query for example, the most requesting clients or "how many queries use EDNS0 and for which sizes?"

It is important to note there are already identified actions to countermeasure DDoS flooding attacks. For example, DNS server implementations allow to set a Response Rate Limit (RRL) [11], and Linux's netfilter can drop ANY type queries. In a broader scope, Best Current Practices RFCs [12], [13] recommend different measures including to deny packets with forged IP addresses, close open resolvers, or authenticate DNS clients by signature. These can have a larger positive impact, but require global coordination and action from several Internet actors, especially network providers.

Despite the existing monitoring tools and standardised best practices, we can affirm that the DDoS problem is not solved yet, and research is still required for implementing tools that prevent making DNS servers unavailable.

III. Nx10Gbps traffic processing frameworks

We can categorise the existing traffic generators according to their base: hardware appliances, FPGA and software. Hardware-based devices, such as those produced by Xena, commonly carry out high-speed generation and are highly accurate to control packet rates and timestamping. They are

focused on benchmarking according to standard methodologies such RFC 2544, RFC 2889 or RFC 3918. Xena also provides scripting interface to control packet generation and capture. However, these devices are not easily affordable given their costs. At their turn, NetFPGA cards are built on open source hardware and software, and are especially designed for research and teaching, being also less expensive than hardware appliances. NetFPGA cards are able to achieve highly-accurate packet generation in terms of rates and inter packet gaps.

Software solutions are based on general-purpose computers and specialized network cards produced by Intel and Melanox, among other providers. Even though these solutions relate to specific hardware, they are more affordable than FPGA-based approaches. Software-based solutions depend on the framework that delivers the packets to the hardware interface. The current version of the standard Linux network framework, the New API (NAPI), is poorly efficient in 10Gbps links. The need to provide an alternative has been largely identified by academic and industrial research. Moreno [14], Gallenmüller et al. [15] have studied the NAPI's characteristics that create limitations and bottlenecks. Example of such characteristics are per-packet management of resources, serialized access to traffic for further analysis on a single point, among others.

Different network frameworks such as DPDK [16], HPCAP [17] PFQ [18], and PF_RING [19], have been developed to overcome these limits. While they implement different strategies to take advantage of modern network interfaces capabilities, we can find some similarities among them. For example, they provide multicore support, they manage memory more efficiently, create direct parallel paths between hardware queues and high-level applications, avoid intermediate copies mapping memory regions for direct access, and process batches or streams instead of single packets.

We can also identify some of their particularities. For example, DPDK [16] provides a set of data plane libraries and drivers. Being mainly developed by Intel, DPDK has a strong focus on specific network interfaces and architectures. DPDK proposes an Environment Abstraction Layer (EAL) that interfaces between the hardware environment and the application relying on it. Its main advantages are its stable status and that it counts with an important support from Intel's 6WIND and other providers.

HPCAP is a promising network engine developed by Víctor Moreno, focusing strongly on packet capture [14]. According to Moreno, HPCAP design took into account goals not considered by other frameworks: accurately timestamping incoming packets, controlling packet capture on non-volatile media, and dropping duplicated packets. HPCAP focuses to Intel 82599 network interfaces, providing a kernel driver related to the Intel's ixgbe. To ease the interface with C user applications, HPCAP provides the M3OMon library framework. However, these tools lack of a mature and stable release.

Nicola Bonelli et al. designed the PFQ [18] network framework, especially optimized for multi-core architectures and multiple hardware queue network interfaces. In contrast to other approaches, PFQ provides a kernel driver that connects against any standard network interface kernel driver, then it has less restrictions in terms of hardware. It relies on parallelism to saturate a 10Gbps link using the Intel ixgbe vanilla driver. It also provides a programming language framework for C, C++ and Haskell, as well as pfq-lang, a functional language designed to process packets at in-kernel early stage.

Luca Deri et al. have developed the PF_RING [19] framework, designed for Intel network interfaces, that improves the rates achieved using the Linux NAPI. It includes a Zero Copy (ZC) framework version, inspired by the predecessor PF_RING Direct NIC Access (DNA), able to achieve line-rate on 10Gbps links. This ZC API provides building blocks to perform zero copy operations across threads and user-space applications. However, the PF_RING ZC's non-free license limits the use of this framework version.

IV. Software-based traffic generators

It is possible to find software-based traffic generators, but we focus here on those especially designed for high performance on 10 Gbps links.

A. General purpose generators

For example, Bonelli et al. designed a generator [20] that, relying on parallelism and a new PF_DIRECT socket for the standard Linux network framework, is able to send 13 million 64-byte packets per second, near to line-rate, and to saturate a 10 Gbps link with 128-byte packets. Concerning PF_RING, the zsend generator is able to achieve line-rate with the zero-copy module. But the non-free license limited our tests. As far as we understand, HPCAP lacks a packet generation tool, documentation on how to build it, or a high-level framework that allows to well format packets.

We can find different DPDK-based traffic generators available, such as Wind River's pktgen-dpdk [21], an accelerated version of Ostinato [22] and MoonGen [4]. At the time of this writing, none of them provided support to generate customized DNS queries. Intel's pktgen was designed to run over DPDK. It is able to send or forward synthetic traffic at Nx10Gbps, and it claims that it can be customized via Lua scripts. However, we have been unable to reproduce traffic from script examples included on the source, and documentation lacks information about how to enhance it. At its turn, Ostinato was built over PCAP and it was not designed to produce 10Gbps traffic. However, there have been efforts to adapt it to DPDK. Ostinato's main particularity is a graphical front-end that aims at making simple to customize the traffic to produce. At the same time, its performance may be impacted by this design choice.

MoonGen is a traffic generator that wraps DPDK and provides network stack control by Lua user scripts. Its authors affirm they designed MoonGen requiring that it should be as flexible as possible, it should be able to saturate 10Gbps links with 64-byte packets, and it must control rates and timestamp packets with a high precision level.

From our point of view, the main MoonGen's feature, is that it provides the user with the ability to manage packets

via Lua scripts. MoonGen relies on LuaJIT [23], a just-in-time compiler, and its Foreign Function Interface (FFI) library that makes it possible for Lua to directly interact with DPDK's C libraries and structures. MoonGen authors chose Lua because Snabb Switch [24] have previously demonstrated that it can be used to process packets at high rates. MoonGen confirms this, not only achieving full wire-rate on a single 10Gbps link, but it also rating 178.5 Mpps at 120Gbps thanks to multiple cores controlling six dual-port network cards. Depending on hardware availability, authors claim that it could scale to 100GbE, considering that multiple cores can handle a single port.

MoonGen combines software and hardware-based methods to mitigate timing issues and control inter-packet gaps. It also takes advantage of hardware timestamping to measure latency under sub-microsecond precision.

Authors have also identified some limitations, mainly given by LuaJIT, which can lead to pause times when collecting garbage. MoonGen disables then the garbage collector for most experiments without risk, because all the packet buffers are handled by DPDK, making them transparent to user scripts.

The lack of a mature and fixed release, detailed documentation and stable example scripts could be considered as the important limitations. However, we have been able to reproduce script examples that can saturate a 10Gbps link with 64-byte packets or with TCP-SYN flooding. Taking into account the high level of abstraction that it provides to control packets, we can consider MoonGen as a sound base to build a DNS flexible query generator.

B. DNS-specific attack generators

It is difficult to find publicly available DNS query generators especially designed to produce flooding attacks. We have been able to evaluate SOP [2], that focuses on DNS reflect-and-amplify attacks. SOP gives to the user the flexibility to generate queries from fixed qnames from static IP addresses until random qnames from varying sources. It takes into account EDNS(0) additional records, allowing to advertise a UDP buffer size, and indicate if the reflector is able to handle DNSSEC security records.

SOP relies on the Linux standard communication socket, and as we show in the following sections, this makes it difficult to fully take advantage of 10Gbps links. We compare our generator with SOP in the following Section.

V. A FLEXIBLE DNS QUERIES GENERATOR BASED ON DPDK+MOONGEN

As stated before, we have chosen MoonGen by the flexibility it provides to control packets generation by high-abstraction level Lua scripts. In our case, we needed to design scripts that create attack-equivalent DNS flooding queries, at the maximum possible rate. These scripts require to be easily modifiable to provide the ability to adapt the generator to new or further kind of attacks. In this section we describe how we have structured the scripts, how we have evaluated them and the results we have achieved.

A. MoonGen user scripts to generate DNS queries

As described in [4], MoonGen user scripts require a master function that configures the running environment and then spawns slave functions linked to a specific core and port. These slave functions actually create the packets and send the traffic. MoonGen's Lua scripts run transmission loops at their core, which deliver packet buffers to memory space. Thanks to DPDK, network interfaces directly access those memory spaces, avoiding the overhead produced by the standard network framework of Linux, as explained in the previous section.

The base of our approach is to take advantage of the MoonGen's ability to format and fill the packets by using calling functions, when allocating buffers in memory. To clarify this, the following code snippet provides an example about the function that create memory pools for DNS queries inside the slave function.

```
1  local mem = memory.createMemPool(function(buf)
2          buf:getDnsPacket(ipv4):fill{
3                  ip4Src=genIPv4AddSource(),
4                  ip4Dst=dnsServerIP,
5                  udpSrc=math.random(1025,65534),
6                  ip4TTL=math.random(10,100),
7                  udpDst=53,
8                  dnsQDCount=1,
9                  dnsARCount=1,
10                 dnsMessageContent=genBody(),
11                 pktLength=packetLen}
12         end)
13
14 local bufs = mem:bufArray(MAX_BURST_SIZE)
```

In other words, mem's createMemPool() calls functions that can fill specific fields in the packet. As stated before, DPDK can process batches of packets. This is related to the bufs variable that creates a buffer, where each packet will be formated and filled by mem's createMemPool. Only when this structure is defined, the main transmission loop is executed, represented in the next simplified code.

```
1 while dpdk.running() do
2          bufs:alloc(packetLen)
3          --offload checksums to NIC
4          bufs:offloadUdpChecksums(ipv4)
5          totalSent = totalSent + queue:send(bufs)
6 end
```

In each iteration of this loop, the buffer allocates in memory a burst of packets of the same specified length. Then, it offloads the checksum calculation to the network interface and finally the hardware queue sends the packet buffer through the port the slave function is linked to.

In the example code snippet listed above, we create packets containing one query and one addition record. One of the most relevant callback functions is in charge of filling the DNS data sections. In this case, the DNS Message Content is affected by the value returned by *genBody()*, responsible for correctly filling the query and additional record sections.

B. Varying packet fields

A flooding query generator requires to fill packets with highly irregular values to produce a traffic pattern difficult to identify. In Section II, we have listed fields related to the

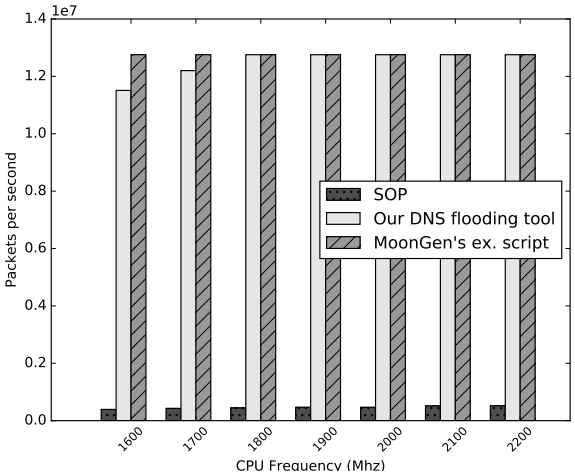

Fig. 1. Packets per second versus CPU frequency. SOP, our DNS query flooding script and MoonGen's tx-multi-core.lua example script.

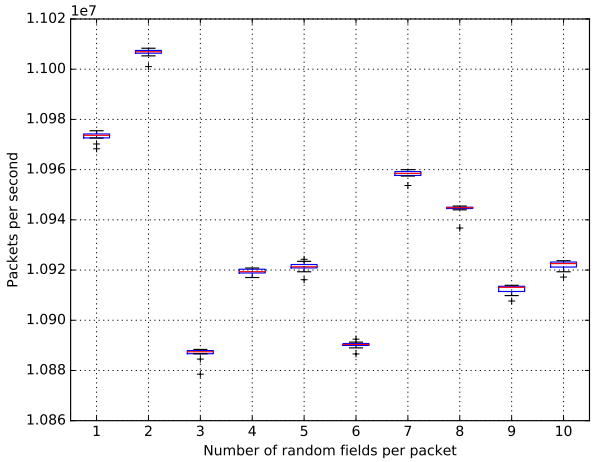

Fig. 2. Packets per second versus number of random fields.

studied attacks that the generator must produce with varying values. If we consider also characteristics inherent to the flow of DNS packets, we can summarize the list of fields as follows: (1) source IP address, (2) source port, (3) packet TTL, (4) DNS query id, (5) queried qualified name, (6) query type and buffer size, and (7) advertised UDP buffer size.

Generating irregular values may have an important impact on performance, which, according to MoonGen authors, depends on the script itself. We can consider two methods to constantly modify these values: using randomizing functions or through arithmetic operators, such as increasing a counter's value. Even if the arithmetic operation method requires less CPU resources, we have chosen the Lua's random() function as a first option for all the fields. Except from the random qname field, the listed values can be easily represented by a single 8-, 16-, or 32-bit integer, and then a single random() call is required. For simplicity, we consider the random qname as a set of 8-bit ASCII characters.

C. Scaling up to Nx10Gbps

Considering that DPDK and MoonGen can distribute the work load of different ports among different CPU cores, it is possible to send traffic through different ports at the same time. As explained in Section IV, MoonGen authors claim it can scale to 100Gbps, so we can optimise the use of our environment and use the maximum number of ports. Our working environment is composed of a single-processor quad-core Intel i7-2600K machine, running Debian Jessie, with 8GB RAM and two Intel network interfaces: a dual SFP+ port X520-DA2 and a dual RJ45 port X520-TA2.

MoonGen requires to reserve a CPU core for the master function. Using the three remaining free cores, our script has been able to saturate three ports in similar conditions between each other, achieving 30Gbps. For synthetic random qnames and reflect-and-amplify query packets, this represents more than 35 million requests per second.

VI. Evaluating performance

In this section, we first assess the impact of CPU frequency on performance, comparing how our script performs against two different tools, running at different CPU frequencies. Then, we evaluate how the number of varying fields affects the performance of our script.

A. Impact of CPU frequency

To evaluate the impact of CPU frequency, we consider the SOP DNS query generator [2] and the MoonGen *tx-multi-core.lua* example script, that generates "empty" packets.

Looking for a "worst case" scenario, we minimized the size of the packets avoiding EDNS additional records, resulting in a 74-byte packets. For SOP and our DNS flooding script, this means patterns of DNS random queries. We configured them to produce queries composed of two labels: an eleven-byte random prefix and a two-byte fixed suffix. While tx-multi-core.lua cannot produce DNS queries, but we have modified it to generate packets of the same length than the other tools.

Figure 1 summarises the outcome of running the three tools to send traffic for 30 seconds for different CPU clock multipliers. First, SOP performed a 392852 pps mean rate at 1.6Ghz and increased up to 518949 pps at 2.2Ghz. Second, MoonGen authors affirm it is able to saturate a 10GbE link with 64-byte packets using a 1.5Ghz core, so we expected full wire-rate for all the tests. Third, our script sent 11.51 Mpps mean rate at 1.6Ghz, 12.2 Mpps nearly line-rate at 1.7Ghz and full line rate from 1.8Ghz. The difference in performance between both scripts, the MoonGen's tx-multi-core example and our DNS generator, can in particular be explained by the table handling operations and the different function calls that correctly format and fill the DNS packets in *genBody()*.

B. Impact of random packets on performance

To evaluate how generating fully random packets impacts performance, we have run our script against different numbers of varying fields. Following a similar 74-byte packet structure

than the previous evaluation, we have set down CPU frequency to 1600Mhz. As Figure 2 shows, variations between generating 1-random and 10-random field packets is not substantial. As shown above, differences in performance disappear when CPU frequency is at least 1800Mhz.

VII. Conclusion

We have studied different software network frameworks and traffic generation tools, and chosen MoonGen to build a DNS-flooding tool, since it provides the highest possible abstraction level. Our implemented tool makes it possible to demonstrate that commodity-hardware can provide a robust and flexible support to simulate malicious traffic against DNS, similar to aggressive DDoS at Nx10Gbps. Our tool is able to saturate high-speed links with ∼70-byte DNS queries on a single Intel i7-2600K CPU core, running at 1800Mhz.

Since attacks evolve fast, taking advantage of different protocol characteristics, the flexibility is a strong requirement for a generator. Our developed tool makes it easy to control the query and additional record sections in DNS packets, allowing to reproduce qrandom and reflect-and-amplify attacks. It can also create fully random packets without impacting transmission rates. This tool strongly relies on the MoonGen generator and, in turn, on Intel DPDK. In consequence, it requires specific network interfaces, although this hardware is more affordable compared to FPGA- or hardware-based solutions. The high-abstraction level provided by MoonGen and the Lua scripts it is based on, guarantees the highest possible level on flexibility. Any change required to, for example, consider a new kind of attack or new extensions to the DNS protocol, can be applied at scripting level. We have also shown that it can scale up to multiples of 10Gbps, especially depending on the number of available cores.

We consider this generator as a useful tool that helps to evaluate strategies to protect DNS servers. 10Gbps-able defense tools are currently needed, so our future work focuses on evaluating the limits of commodity hardware to analyse incoming traffic. While we can identify some similar requirements in tools that generate flood attacks on one side, and tools that detect them on the other, the analysis of traffic addressed to actual DNS servers needs to be highly precise. DNS servers must guarantee to answer to any request, so protection tools must avoid dropping actual queries. This particular requirement challenges general-purpose hardware and the software it relies on. Further work include evaluating if DPDK+MoonGen is still a sound support or if a capture focused-framework could be more suitable.

Acknowledgment

The authors would like to thank the developers of MoonGen, upon whose high-abstraction level and giant shoulders we stand. We also thank Télécom Bretagne student Nicolas Tollenaere, who helped to evaluate the generation tools.

References

[1] "Rapport 2014 sur la résilience de l'Internet français," Agence Nationale de la Sécurité des Systèmes d'Information, Tech. Rep., 2014.

[2] S. Bortzmeyer, "Persée et la Gorgone : attaques par déni de service utilisant le DNS, et les contre- mesures," in Les Journées Réseaux (JRES), 2013. [Online]. Available: http://www.bortzmeyer.org/files/jres2013-dos-article.pdf

[3] R. Dobbins, "State of Danger - talk at AusNog," 2011, last accessed on: February 26th 2016. [Online]. Available: http://www.ausnog.net/sites/default/files/ausnog-05/presentations/ausnog-05-d02p05-roland-dobbins-arbor.pdf

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in Proc. IMC'15, Tokyo, Japan, Oct. 2015.

[5] S. Balakrichenan, "Disturbance in the DNS - talk at DNS-OARC," 2014, last accessed on: February 26th 2016. [Online]. Available: https://indico.dns-oarc.net/event/20/session/3/contribution/37/material/slides/0.pdf

[6] RIPE NCC, "DNS Monitoring Service (DNSMON)," Last accessed on: March 10th 2016. [Online]. Available: https://atlas.ripe.net/dnsmon;https://frama.link/DNSMON-FR-2016-09-04

[7] M. Weinberg, "Review and analysis of attack traffic against A-root and J-root on November 30 and December 1, 2015 - talk at DNS-OARC," 2015. [Online]. Available: https://indico.dns-oarc.net/event/22/session/4/contribution/7/material/slides/0.pptx

[8] J. Damas, M. Graff, and P. Vixie, Extension Mechanisms for DNS (EDNS(0)), IETF, Apr. 2013. [Online]. Available: https://www.ietf.org/rfc/rfc6891.txt

[9] Nagios Enterprises, LLC, "Nagios." [Online]. Available: https://www.nagios.org

[10] AFNIC, "DNSWitness." [Online]. Available: http://www.dnswitness.net

[11] P. Vixie, "DNS Response Rate Limiting - ISC-TN-2012-1-Draft1," ISC, Tech. Rep., Apr. 2012. [Online]. Available: http://ss.vix.su/~vixie/isc-tn-2012-1.txt

[12] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," IETF, Tech. Rep., May 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2827.txt

[13] J. Damas and F. Neves, Preventing Use of Recursive Nameservers in Reflector Attacks, IETF, Oct. 2008. [Online]. Available: https://www.ietf.org/rfc/rfc5358.txt

[14] V. Moreno, "Harnessing low-level tuning in modern architectures for high-performance network monitoring in physical and virtual platforms," Ph.D. dissertation, Universidad Autónoma de Madrid, 2015.

[15] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of Frameworks for High-Performance Packet IO," in Proc. ANCS'15. IEEE Computer Society, 2015, pp. 29–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=2772722.2772729

[16] DPDK, "Data Plane Development Kit," Last accessed on: February 18th 2016. [Online]. Available: http://dpdk.org

[17] V. Moreno, J. Ramos, P. S. del Rio, J. Garcia-Dorado, F.J.Gomez-Arribas, and J.Aracil, "Commodity Packet Capture Engines : Tutorial, Cookbook and Applicability," IEEE Comunications Surveys and Tutorials, 2015.

[18] N. Bonelli, A. D. Pietro, S. Giordano, and G. Procissi, "On Multi-gigabit Packet Capturing with Multi-core Commodity Hardware." in Proc. PAM 2012, vol. 7192. Springer, 2012, pp. 64–73. [Online]. Available: http://dblp.uni-trier.de/db/conf/pam/pam2012.html#BonelliPGP12

[19] PF_RING, "High-speed packet capture, filtering and analysis." Last visited on: February 18th 2016. [Online]. Available: http://www.ntop.org/products/packet-capture/pf_ring

[20] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, "Flexible High Performance Traffic Generation on Commodity Multi—core Platforms," in Proc. TMA'12. Springer-Verlag, 2012, pp. 157–170. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28534-9_17

[21] W. R. Systems, "Pktgen," last accessed on: February 22th 2016. [Online]. Available: http://dpdk.org/browse/apps/pktgen-dpdk/refs

[22] S. P., "DPDK-Accelerated Ostinato prototype," last accessed on: February 19th 2016. [Online]. Available: https://github.com/pstavirs/dpdk-ostinato

[23] LuaJIT, "Lua Just in Time Compiler," last accessed on: February 19th 2016. [Online]. Available: http://luajit.org

[24] L. Gorrie, "Snabb Switch." [Online]. Available: https://snabb.co